2_Vectorizacion

February 7, 2025

Tema 1: Computación Inteligente (LLM)

Vectorización (Embeddings)

Prof. Wladimir Rodríguez

wladimir@ula.ve

Departamento de Computación

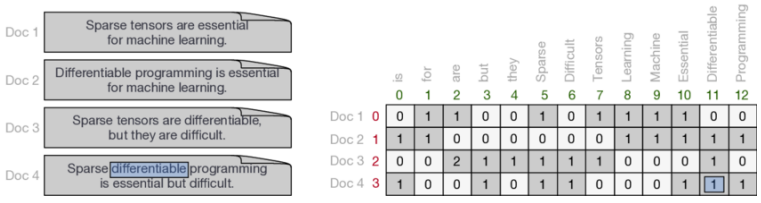
0.1 Motivación - Representaciones

- Problema: Representación de palabras en modelos de lenguaje
  - One-hot encoding: Representa palabras como vectores binarios.
  - Bag of Words (BoW): Representa documentos como conteo de palabras.
  - TF-IDF: Mejora BoW ponderando palabras según importancia.
- Limitaciones:
  - Alta dimensionalidad (matrices dispersas).
  - No captura relaciones semánticas entre palabras.
  - No considera contexto en el que aparece la palabra.

Codificación One-hot

el	1	0	0	0	0	0	0
perro	0	1	0	0	0	0	0
ladra	0	0	1	0	0	0	0
corre	0	0	0	1	0	0	0
en	0	0	0	0	1	0	0
la	0	0	0	0	0	1	0
calle	0	0	0	0	0	0	1

0.2 Bolsa de Palabras



0.3 ¿Qué es la Vectorización (Embeddings)?

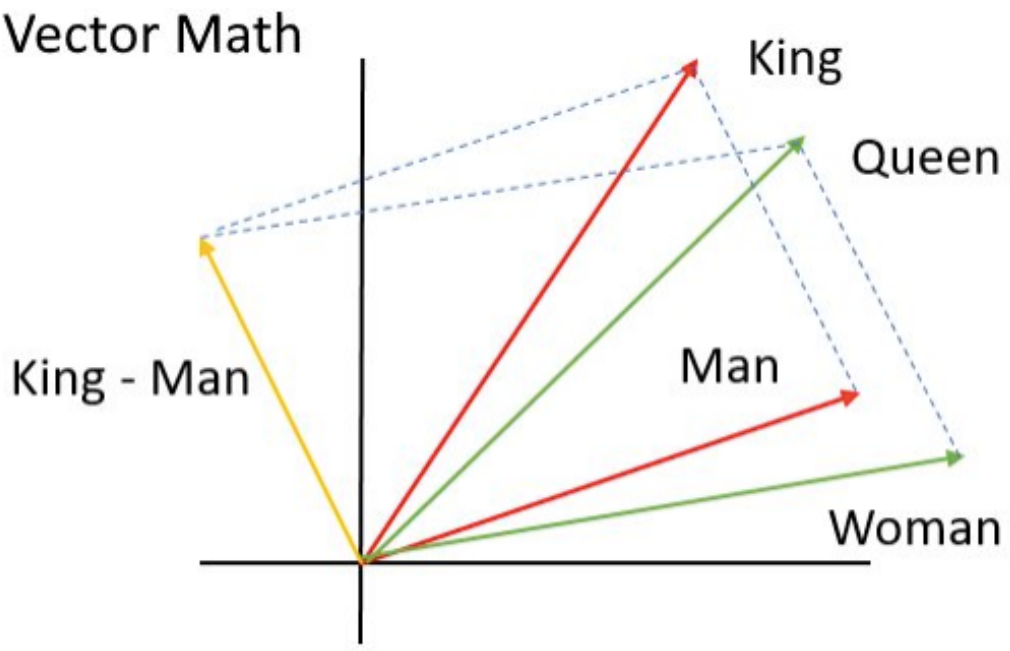
- Definición:
  - Representaciones densas y continuas de palabras en un espacio vectorial.
  - Asignan a cada palabra un vector de n dimensiones.
  - Aprendidos a partir de grandes corpus de texto.
- Beneficios:
  - Reducción de dimensionalidad.
  - Captura relaciones semánticas y sintácticas.

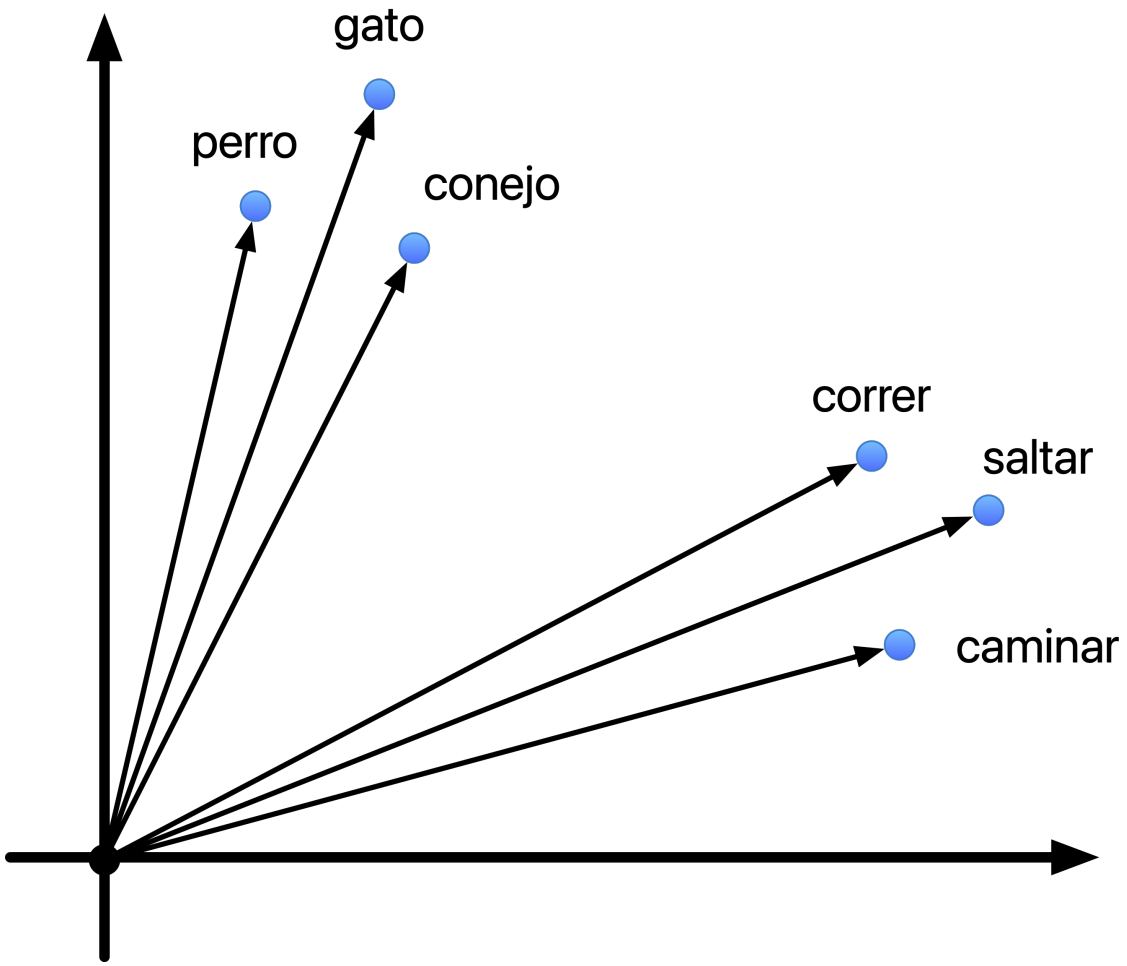
- Permite comparaciones matemáticas entre palabras.

0.4 Relaciones Semánticas y Sintácticas

- Ejemplos de embeddings capturando significado:
  - "Rey - Hombre + Mujer = Reina"
  - Palabras similares en el espacio vectorial están cerca (Ejemplo: "gato", "perro", "mascota").
  - Relaciones morfológicas: "correr" está cerca de "corriendo".

0.5 Relaciones Semánticas





0.6 ¿Cómo se Entrenan los Embeddings?

- Idea principal:
  - Aprender representaciones a partir de contextos de palabras en grandes cantidades de texto.
  - Redes neuronales entrenadas con tareas como predecir palabras en contexto.
- Ejemplo:
  - En "El gato está en la casa", el modelo aprende que "gato" y "casa" aparecen en contextos similares.

0.7 Tipos de Modelos de Embeddings

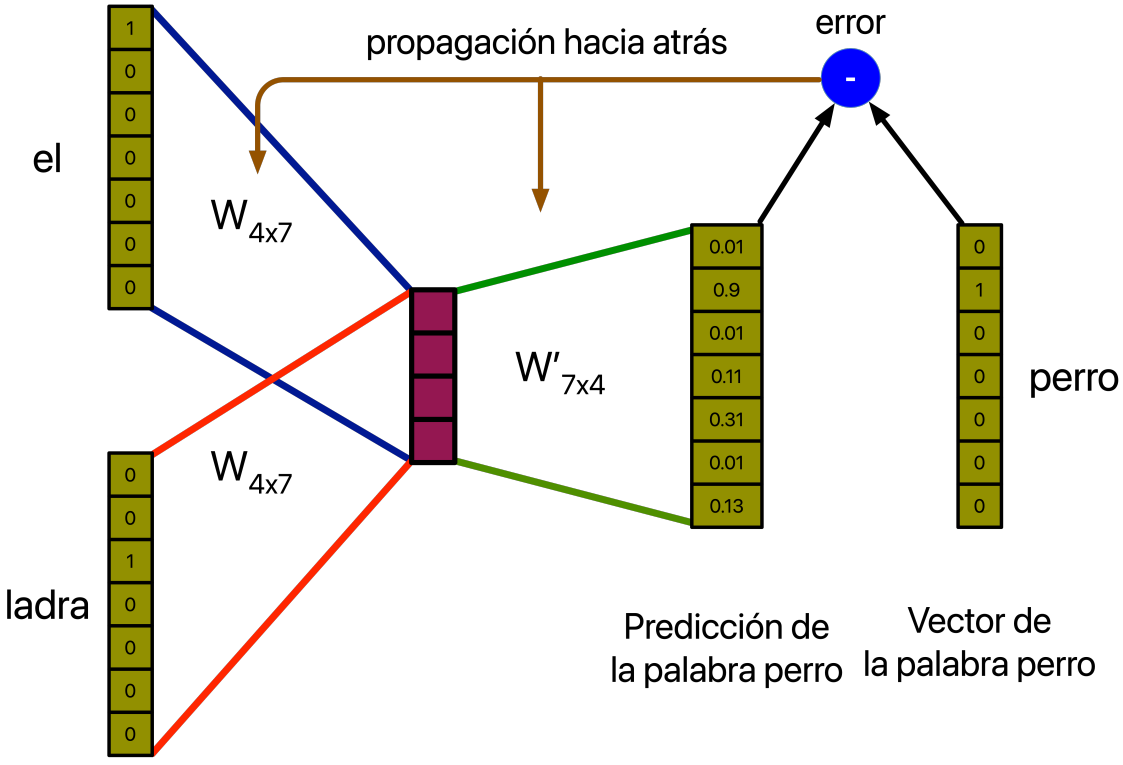
- Embeddings Estáticos (Palabra tiene un único vector fijo):
  - Word2Vec (CBOW y Skip-gram)
  - GloVe
  - FastText

- Embeddings Contextuales (El significado de la palabra cambia según el contexto):
  - ELMo
  - BERT
  - GPT

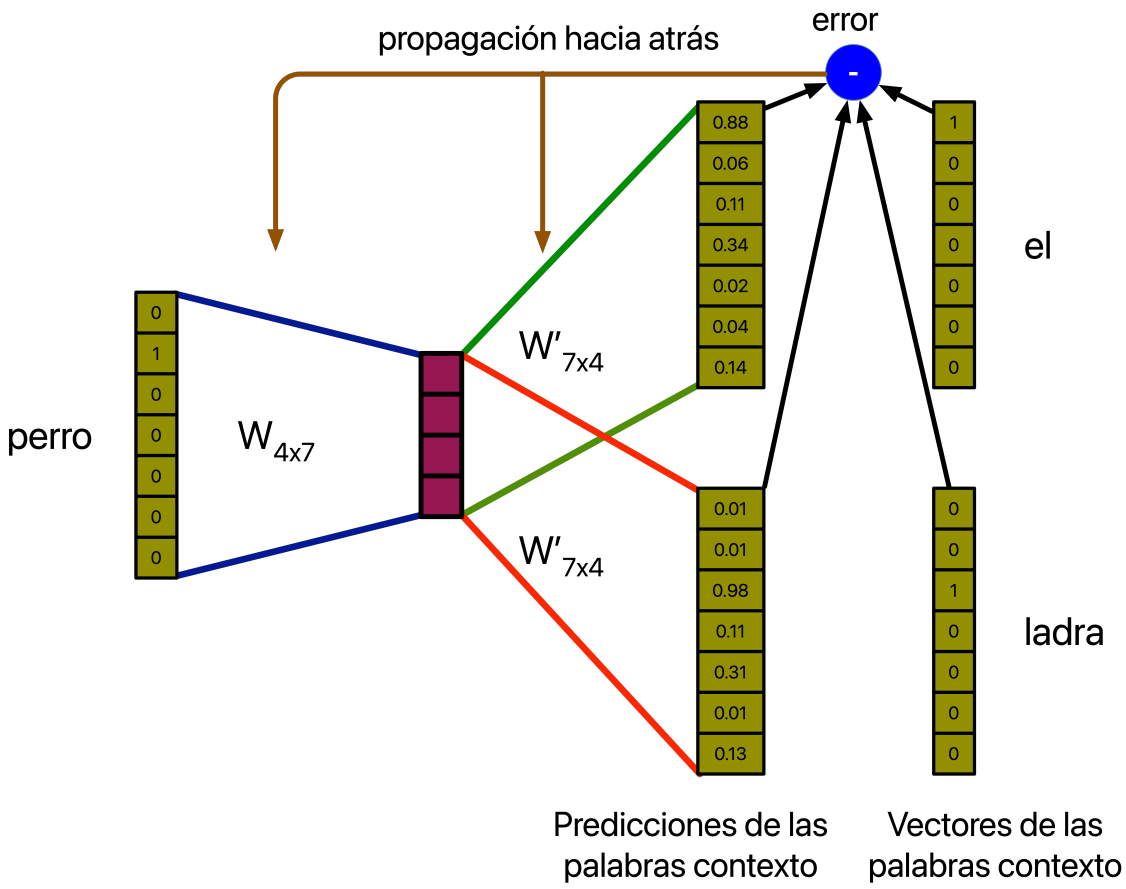
0.8 Modelos Clásicos de Embeddings

- **Word2Vec**
  - Dos enfoques principales:
    - **CBOW (Continuous Bag of Words)**: Predice la palabra central a partir de su contexto.
    - **Skip-gram**: Predice palabras de contexto a partir de una palabra central.
  - Basado en redes neuronales simples.
  - Captura relaciones semánticas bien estructuradas.

0.9 CBOW (Continuous Bag of Words)



0.10 Skip-gram



0.11 Modelos Clásicos de Embeddings

- **GloVe**
  - Basado en la factorización de matrices de coocurrencia.
  - Aprende representaciones a partir de distribuciones globales de palabras.
  - Modela relaciones de palabras de manera efectiva.
- **FastText**
  - Mejora Word2Vec al considerar subpalabras.
  - Captura información morfológica.
  - Mejora representaciones para palabras raras y derivadas.

0.12 Embeddings Contextuales

- Diferencia clave respecto a embeddings estáticos
  - Una misma palabra puede tener diferentes representaciones según el contexto.
  - Modelos más sofisticados permiten capturar significado dinámico.

- **ELMo (Embeddings from Language Models)**
  - Basado en modelos de redes neuronales recurrentes (LSTM bidireccional).
  - Representación dependiente del contexto completo de la oración.

0.13 Embeddings Contextuales

- **BERT (Bidirectional Encoder Representations from Transformers)**
  - Modelo basado en Transformers.
  - Considera contexto anterior y posterior simultáneamente.
  - Usa preentrenamiento con "Máscara de palabras" (Masked Language Model, MLM).
- **GPT (Generative Pre-trained Transformer)**
  - Modelo autoregresivo basado en Transformers.
  - Entrenado para predecir la siguiente palabra en secuencias de texto.
  - Utilizado en tareas de generación de texto.

0.14 Comparación entre Modelos de Embeddings

Modelo	Tipo	Ventaja Principal
Word2Vec	Estático	Representaciones semánticas claras
GloVe	Estático	Basado en coocurrencia global
FastText	Estático	Considera subpalabras
ELMo	Contextual	Representaciones dependientes del contexto
BERT	Contextual	Captura contexto bidireccionalmente
GPT	Contextual	Excelente en generación de texto

0.15 Aplicaciones de los Embeddings

- **Clasificación de Texto**
  - Spam vs. No spam en correos electrónicos.
  - Detección de sentimiento en redes sociales.
- **Búsqueda Semántica y Recuperación de Información**
  - Motores de búsqueda más precisos (Google, Bing).
  - Sistemas de recomendación basados en similitud de palabras y conceptos.

0.16 Aplicaciones de los Embeddings

- **Generación de Texto**
  - Chatbots y asistentes virtuales (Siri, Alexa, ChatGPT).
  - Composición de artículos y resúmenes automáticos.

- **Procesamiento de Texto en Medicina y Finanzas**
  - Análisis de historias clínicas para diagnóstico asistido.
  - Procesamiento de documentos legales y financieros.

0.17 Implementación Práctica en Código

0.17.1 Ejemplo: Uso de Word2Vec en Python**

[1]:
```
from gensim.models import Word2Vec

# Datos de entrenamiento (ejemplo simple)
corpus = [["gato", "perro", "animal"], ["coche", "camión", "vehículo"],␣
 ↪["computadora", "teclado", "tecnología"]]

# Entrenar modelo Word2Vec
modelo = Word2Vec(sentences=corpus, vector_size=10, window=2, min_count=1,␣
 ↪workers=4)

# Obtener el embedding de "gato"
print(modelo.wv["gato"])
```

```
[-0.00856557  0.02826563  0.05401429  0.07052656 -0.05703121  0.0185882
  0.06088864 -0.04798051 -0.03107261  0.0679763 ]
```

0.17.2 Ejemplo: Uso de Embeddings Preentrenados de BERT

[5]:
```
from transformers import BertTokenizer, BertModel
import torch

# Cargar modelo preentrenado
modelo = BertModel.from_pretrained("bert-base-uncased")
tokenizador = BertTokenizer.from_pretrained("bert-base-uncased")

# Tokenizar texto
entrada = tokenizador("Hola, ¿cómo estás?", return_tensors="pt")
print(entrada)
# Obtener embeddings
salida = modelo(**entrada)
print(salida.last_hidden_state.shape)  # Dimensiones del embedding
```

```
{'input_ids': tensor([[  101,  7570,  2721,  1010,  1094, 18609,  9765,  3022,
1029,   102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
torch.Size([1, 10, 768])
```

[ ]: